GUIA desarrolladores

Contents

[Introducción 1](#_Toc436060720)

[Extensibilidad 1](#_Toc436060721)

[Componentes 1](#_Toc436060722)

[Bootstraper 1](#_Toc436060723)

[Listener 1](#_Toc436060724)

[MessageHandler 1](#_Toc436060725)

[Repository 1](#_Toc436060726)

[TESTS 2](#_Toc436060727)

[Tests unitario 2](#_Toc436060728)

[Tests Integración 2](#_Toc436060729)

[Añadir un nuevo messageHandler 2](#_Toc436060730)

# Introducción

Davalor.SynchronizationManager es una aplicación que gestiona la sincronización de los distintos motores de persistencia existentes en davalor mediante el intercambio de mensajes entre los nodos del sistema.

# Extensibilidad

La aplicación se ha desarrollado con la idea de extensibilidad en mente previendo La aparición de nuevas sincronizaciones de datos, por ello el sistema está construido sobre [MEF](https://msdn.microsoft.com/es-es/library/dd460648(v=vs.110).aspx) que permite la rápida adhesión de nuevos componentes que se integraran dentro de la solución preexistente sin necesidad de modificar el código existente.

# Componentes

## Bootstraper

Gestiona la carga de todos los componentes. Su implementación está en la clase Davalor.SynchronizationManager.Host.BootStrapper

## Listener

Escucha los mensajes de la interfaz definida (a día de hoy kafka), cada vez que recibe un mensaje lo deserializa, lo valida y en caso de ser correcto lo pasa al bus interno. Los listener definidos están implementados en el proyecto Davalor.SynchronizationManager.Listener

## MessageHandler

Se subscriben al bus interno de la aplicación a la espera de mensajes. Una vez llegan los mensajes realiza las tareas de sincronización requeridas. Todos los messageHandler deben:

1. Gestionar un único de agregado concreto
2. Heredar de Davalor.SynchronizationManager.Domain.MessageHandling.GenericMessageHandler Al heredar de esta clase conseguimos:
   1. Integrar la gestión de los mensajes en el sistema de trazas.
   2. Una implementación por defecto de tipo CRUD para el agregado.
   3. La capacidad de asociar tipos de mensaje a funciones concretas mediante la propiedad \_handlers
   4. Integrar nuestro handler en el pipeline del servicio.

## Repository

Abstracciones de acceso a los mecanismos de persistencia. Actualmente esta dividido en:

* Factoría (Davalor.SynchronizationManager.Repository.SynchroRepositoryFactory). Crea instancias de repositorios
* Repositorios. Uno por tipo de agregado. Los repositorios si lo desean pueden heredar del repositorio genérico Davalor.SynchronizationManager.Repository.GenericDataService este incluye métodos base para:
  + Save. Guarda el estado del agregado.
  + Update. modifica el estado del agregado.
  + Delete. Elimina el agregado
  + Exists. Inidica si el agregado existe o no.
  + HasBeenChanged. Indica si el agregado ha sido modificado desde la fecha indicada
  + Dispose.
* dbContext. Uno por tipo de agregado y sistema de persistencia. Gestiona los mapeos de los agregados contra el sistema relacional concreto.

# TESTS

## Tests unitarios

## Tests Integración

# Añadir un nuevo messageHandler

Para añadir un nuevo messageHandler es necesario:

1. Crear una clase que herede de GenericMessageHandler indicando el tipo de agregado que va a gestionar.
2. Crear un constructor que:
   1. Llame al constructor de la clase base pasándole: ISynchroRepositoryFactory, IMessageDecrypter, IServiceEvents
   2. Comprobar si el handler está configurado. Ejemplo var config = configuration.MessagesHandlers.FirstOrDefault(cfg => cfg.MessageHandlerName.Equals(\_messageHandlerName, StringComparison.OrdinalIgnoreCase));
   3. Incorporar los sistemas a sincronizar configurados. \_systemsToSyncronize.AddRange(config.SystemToSynchronize);
   4. Asociar los tipos de mensajes deseados a la función que se ha de ejecutar cuando dicho mensaje llegue. Ejemplo: this.\_handlers.Add(typeof(RegisteredLocation), this.Create);
   5. Indicar a la clase base que empiece a escuchar StartListening();
3. Marcar el constructor con el atributo [ImportingConstructor]